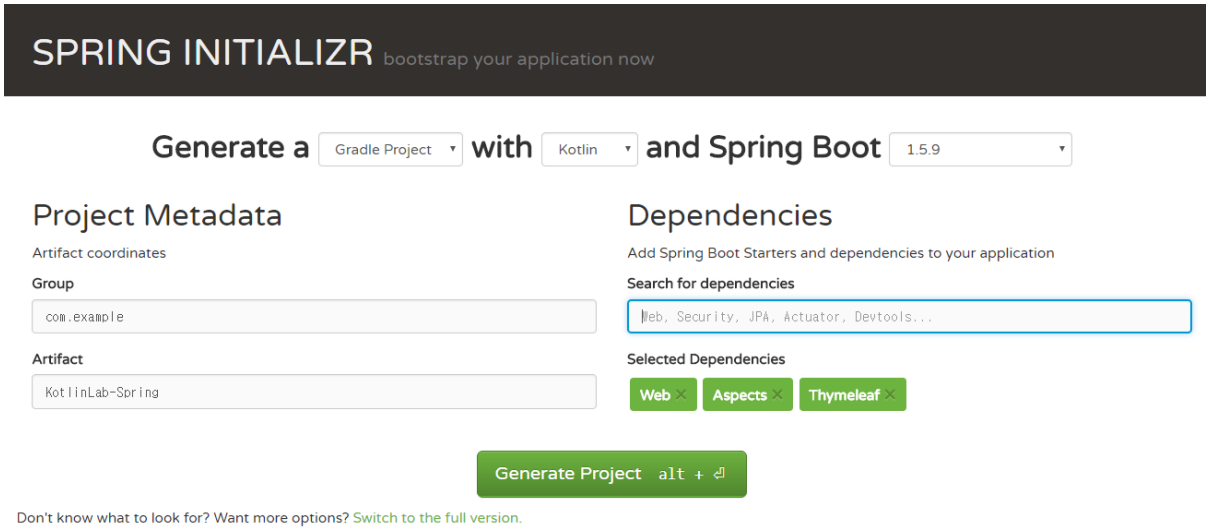
27장. 코틀린으로 스프링 프레임워크 개발

27.1. 스프링 부트로 개발환경 구축하기

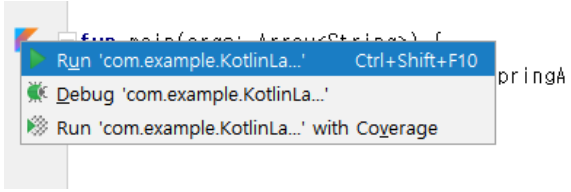


27.1. 스프링 부트로 개발환경 구축하기

```
import org.springframework.boot.SpringApplication
import org.springframework.boot.autoconfigure.SpringBootApplication

@SpringBootApplication
class KotlinLabSpringApplication

fun main(args: Array<String>) {
    SpringApplication.run(KotlinLabSpringApplication::class.java, *args)
}
```

```
import org.springframework.web.bind.annotation.GetMapping
import org.springframework.web.bind.annotation.RequestParam
import org.springframework.web.bind.annotation.RestController
import java.util.concurrent.atomic.AtomicLong

data class Result(val id: Int, val title: String, val content: String)

@RestController
class HelloController {

    @GetMapping("/hello")
    fun hello(@RequestParam(value = "id", defaultValue = "0") id: Int,
              @RequestParam(value = "title", defaultValue = "DefaultTitle") title: String,
              @RequestParam(value = "content", defaultValue = "DefaultContent") content: String) =
            Result(id, title, content)
}
```

27.1. 스프링 부트로 개발환경 구축하기



27.2. 코틀린으로 개발하는 스프링 프로그램

27.2.1. 스프링 IOC – 의존성 주입

- IOC는 Inversion of Control의 약어로 스프링 프레임워크의 핵심
- Bean 클래스를 스프링 프레임워크에 등록하여 필요한 곳에 DI(Dependency Injection) 개념으로 주입
- @Service , @Controller, @Repository 등이 스프링 프레임워크에 Bean 을 등록하기 위한 어노테이션.
- @Autowired을 이용해 객체를 생성, 주입

```
@Service
class IOCTestService {
    fun sayHello(name: String): String{
        return "Hello $name"
    }
}
```

```
@RestController
class IOCTestController {

    @Autowired
    lateinit var service: IOCTestService

    @GetMapping("/ioc")
    fun hello(@RequestParam(value = "name") name: String) = service.sayHello(name)

}
```

27.2. 코틀린으로 개발하는 스프링 프로그램

constructor injection

- DI는 Constructor Injection 과 Setter Injection 두가지 제공

```
@RestController
class IOCTestController2 @Autowired constructor(val service: IOCTestService){

    @GetMapping("/ioc2")
    fun hello(@RequestParam(value = "name") name: String) = service.sayHello(name)

}
```

constructor injection 을 보조생성자에서 이용 가능

```
@RestController
class IOCTestController3 {

    lateinit var service: IOCTestService

    @Autowired constructor(service: IOCTestService){
        this.service=service
    }

    @GetMapping("/ioc3")
    fun hello(@RequestParam(value = "name") name: String) = service.sayHello(name)

}
```

27.2. 코틀린으로 개발하는 스프링 프로그램

setter injection

- 코틀린의 경우는 필드가 아니라 프로퍼티

```
@RestController
class IOCTestController4 {

    @set:Autowired lateinit var service: IOCTestService

    @GetMapping("/ioc4")
    fun hello(@RequestParam(value = "name") name: String) = service.sayHello(name)

}
```

27.2. 코틀린으로 개발하는 스프링 프로그램

27.2.2. 스프링 AOP

- AOP(Aspect Oriented Programming - 관점지향 프로그래밍)
- 관심의 분리(Separation of Concerns)

```
@Service
class AOPTestService {
    fun sayHello(name: String): String{
        return "Hello $name"
    }
}
```

```
@Aspect
@Component
class LoggingAOP {
    @Before("execution(* com.example.KotlinLabSpring.aop.AOPTestService.sayHello(..))")
    fun beforeLogging() {
        println("beforeLogging.......")
    }
}
```

27.2. 코틀린으로 개발하는 스프링 프로그램

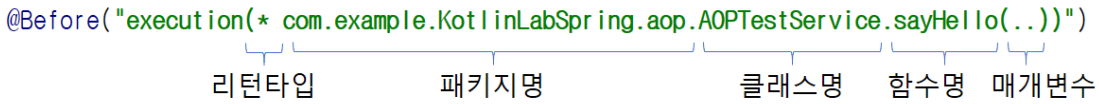
- 스프링 프레임워크에서 AOP 적용은 내부적으로 CGLIB 를 이용
- CGLIB(Code Generator Library)는 런타임에 동적으로 자바 클래스의 프록시를 생성해 주는 기능
- 코틀린의 클래스는 기본이 final
- AOP의 Advice 클래스는 그 자체로 서브 클래스가 동적으로 만들어져 동작함을 목적으로 함으로 개발자가 일일이 open 예약어를 추가해 주어야 한다.
- allopen이 이를 대행

```
dependencies {
      //.....................
       classpath("org.jetbrains.kotlin:kotlin-allopen:${kotlinVersion}")
   }
}

apply plugin: 'kotlin-spring'
```

27.2. 코틀린으로 개발하는 스프링 프로그램

pointcut

- @Before는 Advice가 적용될 특정 시점을 정의하기 위한 어노테이션으로 이를 pointcut 이라고 부른다.




```
@Aspect
@Component
class LoggingAOP {
    @Before("execution(* com.example.KotlinLabSpring.aop.AOPTestService.*(..))")
    fun beforeLogging() {
        println("beforeLogging.......!!!")
    }
    @Before("execution(* com.example.KotlinLabSpring.aop.AOPTestService.*(String))")
    fun beforeLogging2() {
        println("beforeLogging2.......!!!")
    }
}
```

27.2. 코틀린으로 개발하는 스프링 프로그램

- @Before만 있는 것이 아니라 @AfterReturning, @AfterThrowing, @Around 등.

```
@AfterReturning(pointcut = "execution(* com.example.KotlinLabSpring.aop.AOPTestService.*(..))", returning =
"retVal")
fun afterLogging(retVal: Any) {
    println("afterLogging $retVal")
}

@AfterThrowing("execution(* com.example.KotlinLabSpring.aop.AOPTestService.*(..))")
fun afterThrowing() {
    println("afterThrowing....")
}

@Around("execution(* com.example.KotlinLabSpring.aop.AOPTestService.*(String))")
fun around(pjp: ProceedingJoinPoint): Any {
    println("around before....")
    val returnVal= pjp.proceed()
    println("around after....")
    return returnVal
}
```

27.2. 코틀린으로 개발하는 스프링 프로그램

27.2.3. 스프링 MVC

- MVC 모델은 하나의 업무 처리를 Model, View, Controller 로 역할을 분리해서 개발

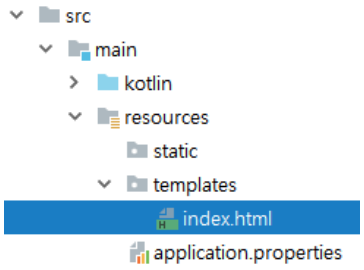
- Model은 비즈니스 업무로직과 업무에 의한 데이터 표현
- View는 화면
- Controller 의 역할은 클라이언트 요청 분석

- 동적 데이터를 html 페이지에 출력하겠다면 템플릿 엔진이 필요하며 thymeleaf는 그중 하나의 엔진

```
dependencies {
    //.....................
    compile('org.springframework.boot:spring-boot-starter-web')
    compile("org.springframework.boot:spring-boot-starter-thymeleaf")
}
```

27.2. 코틀린으로 개발하는 스프링 프로그램

```
import org.springframework.stereotype.Controller
import org.springframework.web.bind.annotation.RequestMapping

@Controller
class MVCController {

    @RequestMapping("/mvc")
    fun doIndexGet(): String{
        return "index"
    }
}
```



27.2. 코틀린으로 개발하는 스프링 프로그램

Thymeleaf 에 의한 동적 데이터 출력

```
data class MyRequestParams(var no: Int = 0 , var name: String = "")
@Controller
class MVCController {

    @RequestMapping("/mvc")
    fun doIndexGet(params: MyRequestParams, model: Model): String{

        model.addAttribute("params", "Hello ${params.no} -  ${params.name}")
        return "index"
    }
}
```

```
<html xmlns:th="http://www.thymeleaf.org">
<head lang="en">
    <meta http-equiv="Content-Type" content="text/html;charset=UTF-8"/>
</head>
<body>
<h1>Hello Spring MVC with Kotlin</h1>

<h3 th:text="${params}"></h3>

</body>
</html>
```

27.2. 코틀린으로 개발하는 스프링 프로그램

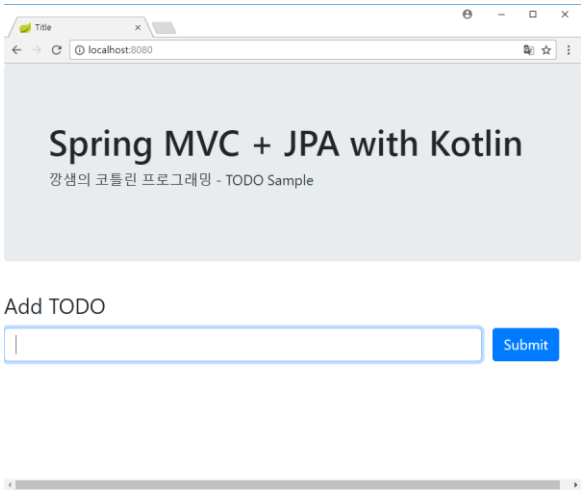
Model 객체 그대로 출력

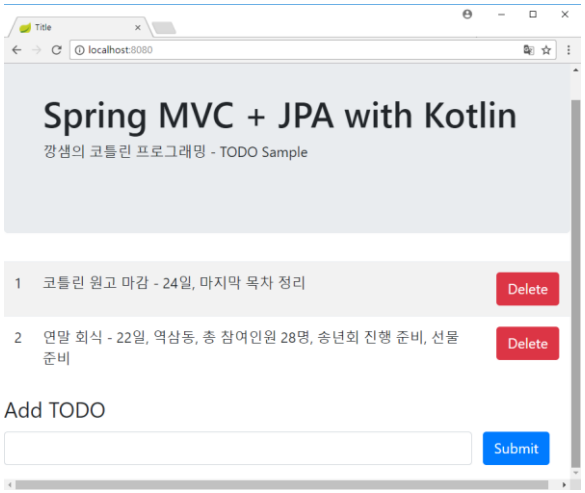
```
data class MyRequestParams(var no: Int = 0 , var name: String = "")
data class MyModelData(var data1: String = "", var data2: String = "")
@Controller
class MVCController {

    @RequestMapping("/mvc")
    fun doIndexGet(params: MyRequestParams, model: Model): String{

        model.addAttribute("params", "Hello ${params.no} -  ${params.name}")
        model.addAttribute("myModel", MyModelData("hello","world"))
        return "index"
    }
}
```

```
<h3 th:text="${params}"></h3>
<table th:object="${myModel}">
    <tr>
        <th>Id</th>
        <th>Name</th>
    </tr>
    <tr>
        <td th:text="*{data1}"></td>
        <td th:text="*{data2}"></td>
    </tr>
</table>
```

27.3. TODO 웹 애플리케이션 개발

27.3.1. 애플리케이션 소개





27.3. TODO 웹 애플리케이션 개발

27.3.2. 사용기술

- 프로그래밍 언어 : 코틀린
- 프레임워크 : Spring Boot

- Web
- Aspect
- JPA
- Themeleaf
- HSQLDB